Chapter 4: Queues

Chapter Objectives

- To learn how to represent a waiting line (queue) and how to use the methods in the Queue interface for insertion (offer and add), removal (remove and poll), and for accessing the element at the front (peek and element)
- To understand how to implement the Queue interface using a single-linked list, a circular array, and a double-linked list
- To understand how to simulate the operation of a physical system that has one or more waiting lines using Queues and random number generators

Queue Abstract Data Type

- A queue can be visualized as a line of customers waiting for service
- New customers are placed at the end of the line.
- The customer at the start of the line will be served the next.
- Queue realizes the first come, first served principle.
- Operating systems use queues for providing services, e.g., print queues.

Interface Queue<E>

Methods	Behavior
E element()	Returns the object at the top of the queue without removing it. . If the queue is empty, returns NoSuchElement exception.
E peek()	Returns the object at the top of the queue without removing it. . If the queue is empty, returns null.
E remove()	Returns the object at the top of the queue and removes it. If the queue is empty, returns NoSuchElement exception.
E poll()	Returns the object at the top of the queue and removes it. If the queue is empty, returns null.
boolean offer(E obj)	Appends item obj at the end of queue. Returns **true** if successful false otherwise.

Class LinkedList **Implements the** Queue **Interface** in JAVA 5.0

- Recall that **LinkedList** implements the **List** interface and provides methods for
  - Inserting and removing elements at either end of a double-linked list
  - Examining elements at the end and the beginning of list (using the size() method and the get() method)
- These will be sufficient for implementing queues
- Starting in Java 5.0, LinkedList implements the Queue interface, so you can declare:
  - Queue<String> name = new LinkedList<String>();

Using a Double-Linked List to Implement the Queue Interface

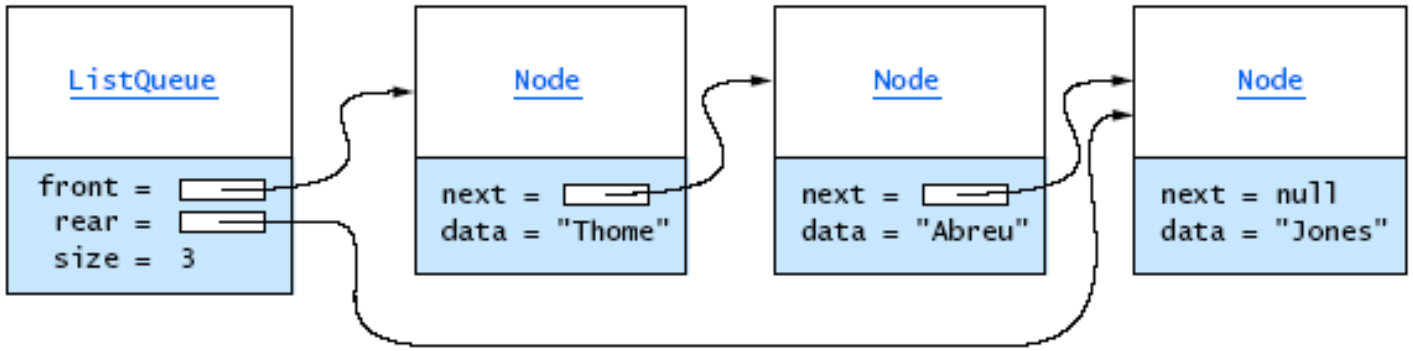
- Insertion and removal at either end of a double-linked list can be done in **O**(1) so either end can be the front (or rear) of the queue
  - Java designers decided to make the head of the linked list the front of the queue.

Using a Single-Linked List to Implement a Queue

- Can implement a queue using a single-linked list
- Class **ListQueue** contains a collection of **Node<E>** objects
- Insertions are at the rear of a queue and removals are from the front
- Need a reference to the last list node

Using a Single-Linked List to Implement a Queue (continued)

FIGURE 6.7
A Queue as a Single-Linked List



Implementing a Queue Using a Circular Array

- List-based implementation of a queue is acceptable in terms of time efficiency but may be wasteful in terms of space.
- Array Implementation
  - Need to know the index of the front, the index of the read, the size, and the capacity.
  - Insertion and removal are both O(1) time.
    - Additional operations on indices required.
    - Doubling of size and copying required for insertion with size equal to capacity.
      - This may not be an issue, because of the “amortized” (or “averaged over time”) effect.

Implementing a Queue Using a Circular Array (continued)

FIGURE 6.8
A Queue Filled with Characters

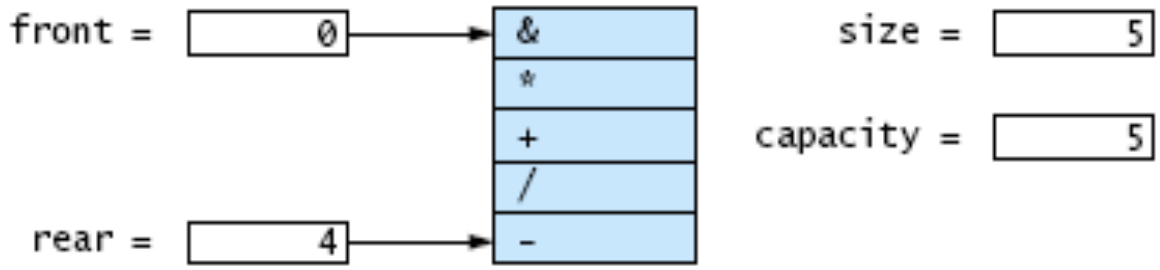
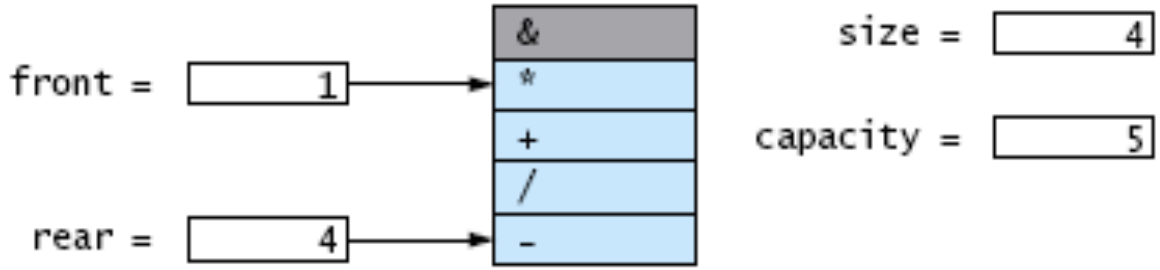
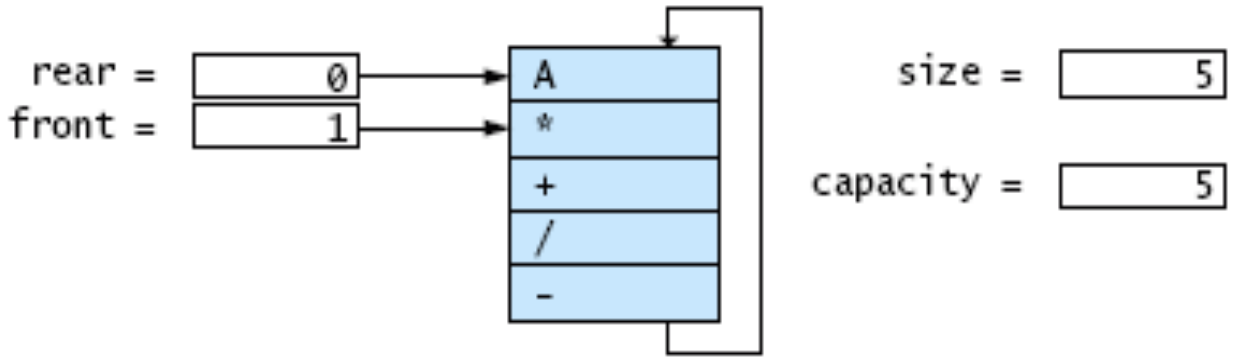


FIGURE 6.9
The Queue after Deletion of the First Element



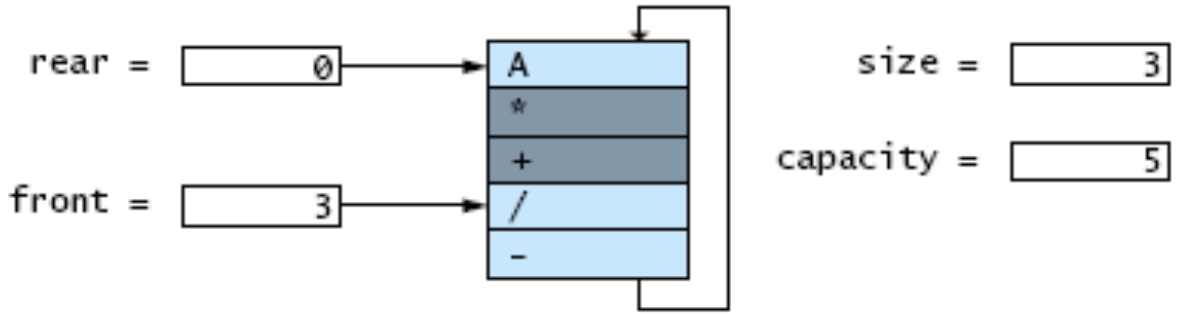
Implementing a Queue Using a Circular Array (continued)

FIGURE 6.10
A Queue as a Circular Array

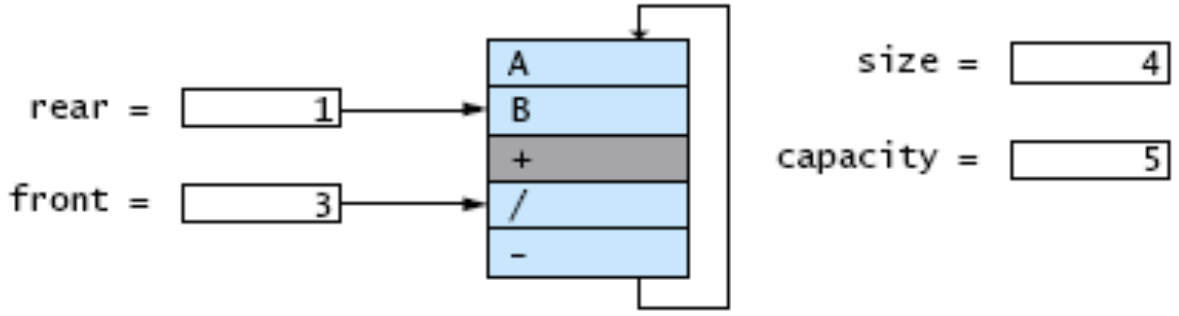


Implementing a Queue Using a Circular Array (continued)

FIGURE 6.11
The Effect of Two Deletions . . .

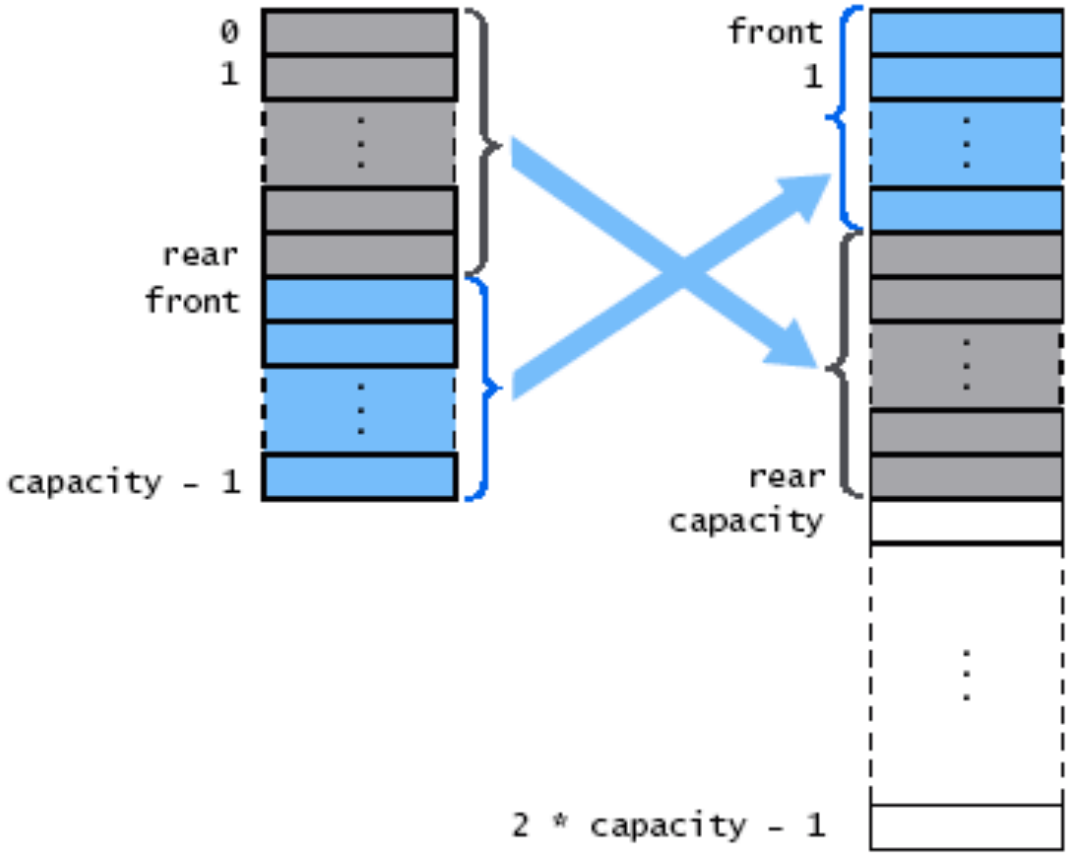


and One Insertion



Implementing a Queue Using a Circular Array (continued)

FIGURE 6.12
Reallocating a Circular Array



Implementing Class ArrayQueue<E>.Iter

- A class that implements Queue must implement all the methods of Collection.
  - Data field **index** stores the subscript of the next element to access
  - The constructor initializes **index** to front when a new Iter object is created
  - Data field **count** keeps track of the number of items accessed so far
  - **Method Iter.remove** throws an Unsupported-OperationException because it would violate the contract for a queue to remove an item other than the first one

Comparing the Three Implementations

- All three implementations are comparable in terms of computation time
- Linked-list implementations require more storage because of the extra space required for the links
  - Each node for a single-linked list would store a total of two references
  - Each node for a double-linked list would store a total of three references
  - A circular array that is filled to capacity would require half the storage of a single-linked list to store the same number of elements … not so bad

Queue with Item Importance

- The position in the queue is determined by importance of item.
- The item with the highest importance is deQueued

Implementation with a Linked List

- Insertion into queue is accomplished by insertion into the list … O(n)
- Heap (to be studied in Chapter 9) accomplishes this in time O(log n)
- There is a better one called Fibonacci Heap, which executes it in time O(log* n)
  - Log*n is the smallest integer k such that the k repeated exponentiation base 2 exceeds n
    - Log*(2) = 1, Log*(16)=2, Log*(65536)=3, Log*(2^65536)=4